Linux Geräte
und
Block E/A

Reinhard Bündgen
buendgen@de.ibm.com

Linux Gerätetypen

- Zeichenorientierte Geräte (character devices)
  - character major & minor Nummern
  - üblicherweise Geräteknoten in /dev
  - mknod Typ c oder u
- Block(orientierte) Geräte (block devices)
  - Block major & minor Nummern
  - Üblicherweise Geräteknoten in /dev
  - mknod Typ b
- Netzwerkgeräte
  - z.B eth0, eth1
  - konfiguriert mit ifconfig oder ip

Major & Minor Nummern

- Spalten 5 und 6 bei `ls -l`
- Werden unabhängig für zeichenorientierte Geräte und Blockgeräte vergeben
- 32bit Wert `kdev_t` (`<linux/kdev_t.h>`)
  - Major: 12 bits `MAJOR(kdev_t dev)`
  - Minor: 20 bits `MINOR(kdev_t dev)`
  - `kdev_t MKDEV(int major, int minor)`
- Können fix oder dynamisch vergeben werden
  - Liste der reservierten Majors und Minors:
    - `Documentation/devices.txt`
  - Liste der registrierten Majors und Minors
    - `/proc/devices`
- Geräteknoten können mit `mknod` im Dateibaum erzeugt werden

Zeichenorientierte Geräte

- Dateityp „c“ (1. Zeichen in `ls -l`)
- Modellierung als Datei
  - `struct file_operations` (`<linux/fs.h>`)
- Spezieller Systemruf `ioctl()`
  - `int ioctl (int fd, unsigned long command, ...)`
- Registirerung von Majors & Minors (`fs/char_dev.c`)
  - Fix: `int register_chrdev_region(dev_t from, unsigned count, const char *name)`
  - Dynamisch: `int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count, const char *name)`
- Struktur für zeichenorientierte Geräte (`<linux/cdev.h>`)
  - `struct cdev (members: file operations, major&minor)`
  - `struct cdev* cdev_alloc(),`
  - `void cdev_init(struct cdev* chrdev, struct file_operations *ops),`
  - `void cdev_add(struct cdev* chrdev, dev_t dev, unsigned count)`

Blockgeräte

- Dateityp „b“ (1. Zeichen in `ls -l`)
- Standardannahme:
  - enthält (montierbares) Dateisystem oder Swapplatz
- Registrierung von Majors (`block/genhd.c`)
  - `int register_blkdev(unsigned int major, const char* name);`
  - `void unregister_blkdev(unsigned int major, const char* name);`
- Generisches Plattenobjekt (`<linux/genhd.h>`)
  - `struct gendisk`
  - `alloc_disk()`, `add_disk()`, `del_gendisk()`
- Blockgeräteoperationen (`<linux/blkdev.h>`):
  - `struct block_device_operations`
  - open, release, ioctl, ...

Netzwerkgeräte

- Keine Darstellung durch Majors&Minors
- Keine Darstellung im Dateisystem
- Zugriff vom Nutzerraum auf NW-Gerät
  - Klassisch (`ifconfig`)
    - `socket()` Systemruf gibt `fd` zurück dann
    - `ioctl(fd,...)` um NW-Gerät zu konfigurieren
  - Neu (`ip`)
    - `fd=socket(PF_NETLINK, …)`
    - Send/receive/ioctl auf `fd` um NW-Gerät zu konfigurieren
- Mehr zu Netzwerkgeräten in LDD3 Kapitel 17

Linux Geräte Modell

- Kernobjekte `struct kobject` (`struct ktype`, `struct kset`)
  - modellieren Geräte - Controller - Bus Hierarchien
  - sind u.a. in Gerätestrukturen (z.B. `struct cdev`) eingebettet
  - siehe `include/linux/kobject.h`
- Kernobjekthierarchie wird im sysfs (`/sys`) abgebildet
  - Verzeichnis beschreibt Kernobjekt
  - ein Datei per Kernelobjektattribut
  - verschiedene „Perspektiven“ auf Geräte (bus, class, device, ...)
- Kernel Event Layer (uevents)
  - `int kobject_uevent(struct kobject * kobj, enum kobject_action action)`
  - schickt Ereignismeldungen von Kern an User Space (z.B. hot plug event)
  - `kobj`: Quelle (extern über sysfs Pfad repräsentiert)
  - `action`: z.B. `KOBJ_ONLINE`, `KOBJ_OFFLINE`, `KOBJ_CHANGE`
  - Kommunikationspfad: netlink (high speed multicast socket)

Blockgerätecharakteristiken

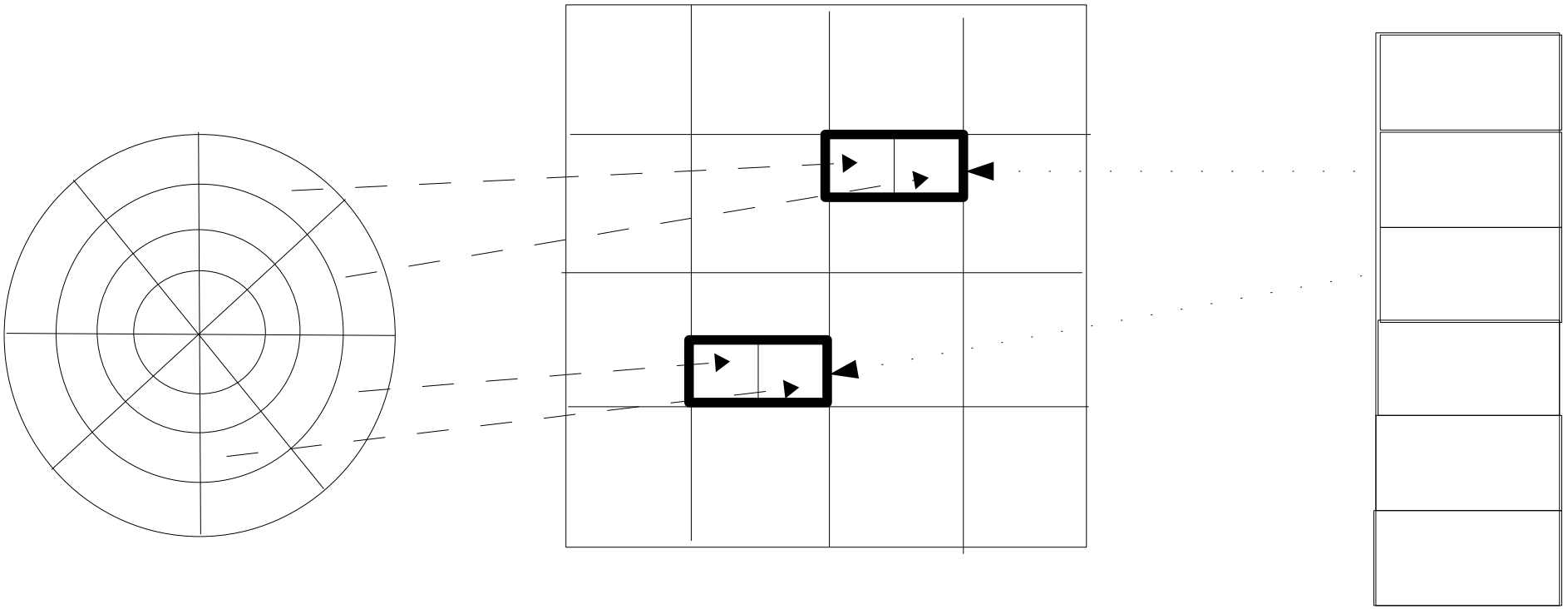
- z.B. Platten, CD/DVD Laufwerke, USB-Sticks, ...
- Kleinste Zugriffseinheit: HW: Sektor (SW: Block)
- Feste Anzahl von Sektoren
- Random Access Zugriff („DASD“)
  - Beliebige Reihenfolge
  - Beliebig oft
  - Lesen und Schreiben beliebig abwechseln
- I.A. langsam
  - Caching von Sektoren im Hauptspeicher: page cache

E/A Konzepte

- Buffer
  - Kopie, von Daten die auf Blockgerät liegen (sollen): „Cache“
- Block E/A Operation
  - Beschreibung einer Datenbewegung zwischen Hauptspeicher (Buffern) und Gerät (Sektoren)
- E/A Anfrage (Request)
  - eine Menge von E/A Operationen deren zusammenhängender Ablauf geplant ist/wird

Block E/A: Begriffe

Platten: Sektoren, Hauptspeicher: Seiten, Dateien: Blöcke



$|Block| = n * |Sektor| = 2^k \ \wedge \ |Block| <= |Seite|$

Puffer (buffers)

- Jeder Sektor wird durch einen buffer repräsentiert
- Buffer head
  - Daten die Buffer beschreiben
  - Typ: `struct buffer_head` in `<linux/buffer_head.h>`
  - Assozeiert Gerät/Sektor mit Seite/Position/Puffergröße
  - Zustand (`b_state` Komponente):
    - `BH_Uptodate` – enthält gültige Daten
    - `BH_Dirty` – Block aktueller als Gerät
    - `BH_Lock` – während E/A
    - `BH_Req` – gehört zu E/A Request
    - `BH_Mapped` – entspricht Daten auf Gerät
    - `BH_New` – neu, noch kein Zugriff
    - `BH_Async_Read` – an asynch read beteiligt
    - `BH_Async_Write` – an asynch write beteiligt
    - `BH_Delay` – hat noch keine entsprechende Sektoren
    - ...

Block E/A Operationen

- Lesen oder Schreiben zusammenhängender Sektoren
- wird durch `bio` Struktur repräsentiert
  - `<linux/blktypes.h>`: `struct bio`
    - Position auf Gerät
    - Operation: Lesen oder Schreiben
    - Liste von Segmenten
      - Segment (`struct bio_vec`)
        - Kontinuierlicher Datenbereich (in einer Seite)
    - Zeiger in aktuelles Segment
- Scatter/Gather Liste

Requests

- `<linux/blkdev.h>`
- Request
  - Beschreibt Zugriff auf zusammenhängende Menge von Sektoren
  - `struct request`
  - Enthält ein oder mehrere bio Strukturen
- Request Queue
  - Beschreibt eine Liste von Requests für ein Gerät
  - `struct request_queue`

E/A Planer (I/O Scheduler)

- E/A ist langsam
  - Insbesondere seek-Operationen
  - Positionierung des Plattenkopfs
  - Dauert mehrere Millisekunden
- Aufgabe:
  - Minimierung von seek-Operationen
- Mittel
  - merging & sorting
- Resultat
  - E/A Planer virtualisiert Platte für mehrere E/A Requests
  - (Leicht) unfair
  - Größerer globaler(!) Durchsatz

Aufzugs Problem

- Betrachte Aufzug im Empire State Buildung
- Ziel
  - fahre eine möglichst geringe Stecke
- Mittel
  - Sortiere die Anfragen
    - Mache in einer Richtung möglichst viele Stops
  - Merge
    - Nehme möglichst viele Personen pro Station mit
- „Lösung“
  - Fahre immer zu nächsten anfragenden Station
- Problem
  - Stockwerke können „verhungern“

Linus Elevator

- Veraltet, aber prototypisch
  - War Standard I/O Scheduler in Linux 2.4
- Sorting:
  - Request Queue nach Sektornummer sortiert
- Merging
  - Direkt aneinander anschließende Requests werden zu einem Request vereinigt
  - Front merging & back merging
- Fairness:
  - Falls es zu alte Requests gibt werden neue (nicht merge-bare) Requests ans Ende der Request Queue gehängt

Lese vs Schreibzugriffe

- Lesezugriffe
  - Applikation muss auf Abschluss der Leseoperation warten
  - Oft hängt eine Leseoperation von einer anderen ab
  - Latenz von Leseoperationen beeinflusst Applikationsperformanz
- Schreibzugriffe
  - Können asynchron von Applikation abgearbeitet werden
    - Ausnahme sync Funktionen
  - Schreiblatenz beeinflusst Applikationsperformanz nur wenig
  - Regelmäßiges Abspeichern verringert das Risiko von Datenverlusten

Deadline I/O Scheduler (1)

- `block/deadline-iosched.c`
- 3 queues:
  - Sorted queue
  - Read FIFO
  - Write FIFO
- Jeder neue Request wird
  - bzgl der Platten Position ge-merge-t & sortiert in die sorted queue eingefügt und
  - je nach Zugriffstyp an die Read- oder Write FIFO gehängt
- Jeder Request wir mit einer Expirationszeit markiert
  - Read request default: 500 ms
  - Write request default 5 sec

Deadline I/O Scheduler (2)

- Abarbeiten des nächsten Requests:
  - Bearbeite den nächsten abgelaufenen Request aus den FIFO Queues
  - Wenn keine abgelaufenen Requests vorhanden, bearbeite den nächsten request aus der Sorted Queue
- Front Merging optional
- Achtung
  - Der Deadline Scheduler gibt keine Garantien darüber, dass ein Request vor Expirationszeit durchgeführt wird

Anticipatory I/O Scheduler

- `block/as-iosched.c`
- DL I/O Scheduler + antizipatorische Heuristik:
  - Nachdem ein Request bearbeitet wird
    - Wird *nicht* der Nächste gesucht,
    - *Sondern* es wird einige ms gewartet, ob nicht ein Request für einen Nachbarsektor abgesetzt wird.
  - Versucht via Statistiken Applikationsverhalten zu erahnen.
  - nicht mehr im aktuellen Kern

Weitere I/O Scheduler

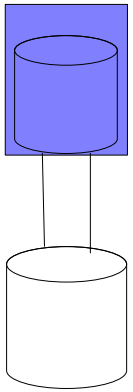
- Complete-Fair-Queuing-I/O Scheduler (CFQ)
  - `block/cfq-iosched.c`
  - eine sortierte und ge-merge-te Queue pro Prozess
  - Round-Robin über Queues: konfigurierbare Anzahl von Requests pro Queue wird bearbeitet
  - Anwendung: Multimedia – Fairness unter Anwendungen
- Noop-I/O Scheduler
  - `drivers/block/noop-iosched.c`
  - keine Requestsortierung, nur Merging
  - Anwendung: Geräte ohne Suchaufwand (z.B. Flash-Memory Karten)

Virtuelle Blockgeräte

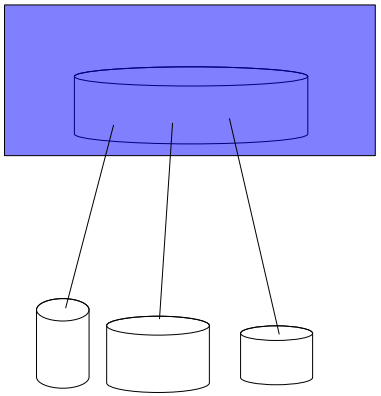
- Physische Blockgeräte:
  - z.B. Einzelne Festplatten (z.B. LUN/WWPN in SAN)
- Virtuelle Blockgeräte
  - Greifen auf Daten anderer (virtueller) Blockgeräte => Stapel von Blockgerätetreibern
  - Partitionen
    - Vom Gerätetreiber verwaltete Aufteilung der Platte
  - Logische Volumes
    - Von Logischen Volume Manger (LVM) verwaltete virtuelle Platte, bestehend aus Sektoren einer oder mehere Platten/Partitionen
  - Multipath Geräte
  - Software RAID (redundant array of independent disks)
    - 0: Striping,1: Spiegelung, 5: ECC
    - `drivers/md`: MD, device mapper
  - Entfernte (remote) Volumes
    - Netzwerk Blockgeräte Treiber: NBD (`drivers/block/nbd.c`)
    - Plattenreplikation für Disaster Recovery Lösungen. DRBD

Virtuelle Blockgeräte

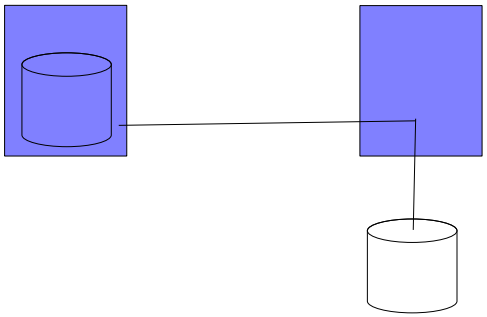
Multipath device



LVM/RAID



Netzwerkblockgerät



Plattenreplikation (DRBD)